##### shiro第二天 shiro授权 shiro和企业项目整合开发

# 复习

什么是权限管理？

权限管理是系统的安全范畴，要求必须是合法的用户才可以访问系统（用户认证），且必须具有该 资源的访问权限才可以访问该 资源（授权）。

认证：对用户合法身份的校验，要求必须是合法的用户才可以访问系统。

授权：访问控制，必须具有该 资源的访问权限才可以访问该 资源。

权限模型：标准权限数据模型包括 ：用户、角色、权限（包括资源和权限）、用户角色关系、角色权限关系。

权限分配：通过UI界面方便给用户分配权限，对上边权限模型进行增、删、改、查操作。

权限控制：

基于角色的权限控制：根据角色判断是否有操作权限，因为角色的变化 性较高，如果角色修改需要修改控制代码，系统可扩展性不强。

基于资源的权限控制：根据资源权限判断是否有操作权限，因为资源较为固定，如果角色修改或角色中权限修改不需要修改控制代码，使用此方法系统可维护性很强。建议使用。

权限管理的解决方案：

* 对于粗颗粒权限管理，建议在系统架构层面去解决，写系统架构级别统一代码（基础代码）。

粗颗粒权限：比如对系统的url、菜单、jsp页面、页面上按钮、类方法进行权限管理，即对资源类型进行权限管理。

* 对于细颗粒权限管理：

粗颗粒权限：比如用户id为001的用户信息（资源实例）、类型为t01的商品信息（资源实例），对资源实例进行权限管理，理解对数据级别的权限管理。

细颗粒权限管理是系统的业务逻辑，业务逻辑代码不方便抽取统一代码，建议在系统业务层进行处理。

基于url的权限管理（掌握）：

企业开发常用的方法，使用web应用中filter来实现，用户请求url，通过filter拦截，判断用户身份是否合法（用户认证），判断请求的地址是否是用户权限范围内的url(授权)。

shiro：

shiro是一个权限管理框架，是apache下的开源项目。相比spring security框架更简单灵活，spring security对spring依赖较强。shiro可以实现web系统、c/s、分布式等系统 权限管理。

shiro认证流程：（掌握）

1、subject(主体)请求认证，调用subject.login(token)

2、SecurityManager (安全管理器)执行认证

3、SecurityManager通过ModularRealmAuthenticator进行认证。

4、ModularRealmAuthenticator将token传给realm，realm根据token中用户信息从数据库查询用户信息（包括身份和凭证）

5、realm如果查询不到用户给ModularRealmAuthenticator返回null，ModularRealmAuthenticator抛出异常（用户不存在）

6、realm如果查询到用户给ModularRealmAuthenticator返回AuthenticationInfo(认证信息)

7、ModularRealmAuthenticator拿着AuthenticationInfo(认证信息)去进行凭证（密码 ）比对。如果一致则认证通过，如果不致抛出异常（凭证错误）。

subject：主体

Authenticator：认证器（ shiro提供）

realm（一般需要自定义）：相当于数据源，认证器需要realm从数据源查询用户身份信息及权限信息。

# 课程安排

1、shiro授权

通过测试代码讲解基于角色权限控制，基于资源的权限控制（掌握）

2、shiro与spring进行整合

项目框架springmvc+mybatis+shiro

3、在整合工程下学习：

认证

授权

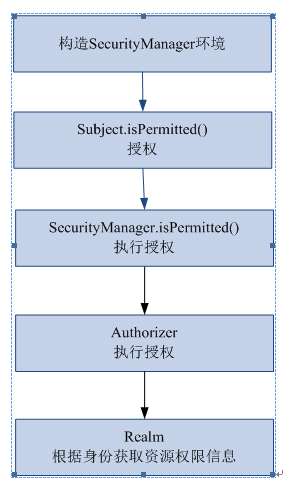
sessionManager

缓存管理

....

# shiro授权

## 授权流程



## 三种授权方法

Shiro 支持三种方式的授权：

* 编程式：通过写if/else 授权代码块完成：

Subject subject = SecurityUtils.getSubject();

if(subject.hasRole(“admin”)) {

//有权限

} else {

//无权限

}

* 注解式：通过在执行的Java方法上放置相应的注解完成：

@RequiresRoles("admin")

public void hello() {

//有权限

}

* JSP/GSP 标签：在JSP/GSP 页面通过相应的标签完成：

<shiro:hasRole name="admin">

<!— 有权限—>

</shiro:hasRole>

## shiro-permission.ini

shiro-permission.ini里边的内容相当于在数据库。

#用户

[users]

#用户zhang的密码是123，此用户具有role1和role2两个角色

zhang=123,role1,role2

wang=123,role2

#权限

[roles]

#角色role1对资源user拥有create、update权限

role1=user:create,user:update

#角色role2对资源user拥有create、delete权限

role2=user:create,user:delete

#角色role3对资源user拥有create权限

role3=user:create

**权限标识符号规则：**资源:操作:实例(中间使用半角:分隔)

user：create:01 表示对用户资源的01实例进行create操作。

user:create：表示对用户资源进行create操作，相当于user:create:\*，对所有用户资源实例进行create操作。

user：\*：01 表示对用户资源实例01进行所有操作。

## 程序编写

*// 基于角色的授权  
// hasRole传入角色标识***boolean** ishasRole = subject.hasRole(**"role1"**);  
System.***out***.println(**"单个角色(role1)判断"** + ishasRole);  
*// hasAllRoles是否拥有多个角色***boolean** hasAllRoles = subject.hasAllRoles(Arrays.*asList*(**"role1"**, **"role2"**, **"role3"**));  
System.***out***.println(**"多个角色(role1、role2、role3)判断"** + hasAllRoles);  
  
*// 使用check方法进行授权，如果授权不通过会抛出异常  
// subject.checkRole("role13");  
  
// 基于资源的授权  
// isPermitted传入权限标识符***boolean** isPermitted = subject.isPermitted(**"user:create:1"**);  
System.***out***.println(**"单个权限判断(user:create:1)"** + isPermitted);  
  
**boolean** isPermittedAll = subject.isPermittedAll(**"user:create:1"**, **"user:delete"**);  
System.***out***.println(**"多个权限判断"** + isPermittedAll);  
  
*// 使用check方法进行授权，如果授权不通过会抛出异常  
// subject.checkPermission("items:create:1");*subject.checkPermission(**"user:create"**);

# 自定义realm进行授权

## 需求

上边的程序通过shiro-permission.ini对权限信息进行静态配置，实际开发中从数据库中获取权限数据。就需要自定义realm，由realm从数据库查询权限数据。

realm根据用户身份查询权限数据，将权限数据返回给authorizer（授权器）。

## 自定义realm

在原来自定义的realm中，修改doGetAuthorizationInfo方法。

*// 用于授权*@Override  
**protected** AuthorizationInfo doGetAuthorizationInfo(  
 PrincipalCollection principals) {  
 *//从 principals获取主身份信息  
 //将getPrimaryPrincipal方法返回值转为真实身份类型（在上边的doGetAuthenticationInfo认证通过填充到SimpleAuthenticationInfo中身份类型），* String userCode = (String) principals.getPrimaryPrincipal();  
  
 *//根据身份信息获取权限信息  
 //连接数据库...  
 //模拟从数据库获取到数据* List<String> permissions = **new** ArrayList<>();  
 permissions.add(**"user:create"**);*//用户的创建* permissions.add(**"items:add"**);*//商品添加权限  
 //....  
  
 //查到权限数据，返回授权信息(要包括上边的permissions)* SimpleAuthorizationInfo simpleAuthorizationInfo = **new** SimpleAuthorizationInfo();  
 *//将上边查询到授权信息填充到simpleAuthorizationInfo对象中* simpleAuthorizationInfo.addStringPermissions(permissions);  
  
 **return** simpleAuthorizationInfo;  
}

## shiro-realm.ini

在shiro-realm.ini中配置自定义的realm，将realm设置到securityManager中。

[main]

#自定义 realm

customRealm=cn.itcast.shiro.realm.CustomRealm

#将realm设置到securityManager，相当 于spring中注入

securityManager.realms=$customRealm

## 测试程序

*// 自定义realm进行资源授权测试*@Test  
**public void** testAuthorizationCustomRealm() {  
 *// 创建SecurityManager工厂* Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(  
 **"classpath:shiro-realm.ini"**);  
  
 *// 创建SecurityManager* SecurityManager securityManager = factory.getInstance();  
  
 *// 将SecurityManager设置到系统运行环境，和spring后将SecurityManager配置spring容器中，一般单例管理* SecurityUtils.*setSecurityManager*(securityManager);  
  
 *// 创建subject* Subject subject = SecurityUtils.*getSubject*();  
  
 *// 创建token令牌* UsernamePasswordToken token = **new** UsernamePasswordToken(**"zhangsan"**,  
 **"111111"**);  
  
 *// 执行认证* **try** {  
 subject.login(token);  
 } **catch** (AuthenticationException e) {  
 e.printStackTrace();  
 }  
  
 System.***out***.println(**"认证状态："** + subject.isAuthenticated());  
 *// 认证通过后执行授权  
  
 // 基于资源的授权，调用isPermitted方法会调用CustomRealm从数据库查询正确权限数据  
 // isPermitted传入权限标识符，判断user:create:1是否在CustomRealm查询到权限数据之内* **boolean** isPermitted = subject.isPermitted(**"user:create:1"**);  
 System.***out***.println(**"单个权限判断"** + isPermitted);  
  
 **boolean** isPermittedAll = subject.isPermittedAll(**"user:create:1"**,  
 **"user:delete"**);  
 System.***out***.println(**"多个权限判断"** + isPermittedAll);  
  
 *// 使用check方法进行授权，如果授权不通过会抛出异常* subject.checkPermission(**"items:add:1"**);  
}

## 授权流程

1、对subject进行授权，调用方法isPermitted（"permission串"）

2、SecurityManager执行授权，通过ModularRealmAuthorizer执行授权

3、ModularRealmAuthorizer执行realm（自定义的CustomRealm）从数据库查询权限数据

调用realm的授权方法：doGetAuthorizationInfo

4、realm从数据库查询权限数据，返回ModularRealmAuthorizer

5、ModularRealmAuthorizer调用PermissionResolver进行权限串比对

6、如果比对后，isPermitted中"permission串"在realm查询到权限数据中，说明用户访问permission串有权限，否则 没有权限，抛出异常。

# shiro与项目整合

## 需求

将原来基于url的工程改成使用shiro实现。

## 创建新工程

创建web工程 ：permission\_shiro1110

## 去除原工程的认证和授权的拦截

删除springmvc.xml中：

<mvc:interceptors>

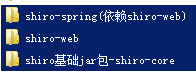
## jar包

包括：

shiro-web的jar、

shiro-spring的jar

shiro-code的jar



## web.xml中配置shiro的filter

在web系统中，shiro也通过filter进行拦截。filter拦截后将操作权交给spring中配置的filterChain（过滤链）

shiro提供很多filter。

在web.xml中配置filter

*<!-- shiro的filter -->  
<!-- shiro过虑器，DelegatingFilterProxy通过代理模式将spring容器中的bean和filter关联起来 -->*<**filter**>  
 <**filter-name**>shiroFilter</**filter-name**>  
 <**filter-class**>org.springframework.web.filter.DelegatingFilterProxy</**filter-class**>  
 *<!-- 设置true由servlet容器控制filter的生命周期 -->* <**init-param**>  
 <**param-name**>targetFilterLifecycle</**param-name**>  
 <**param-value**>true</**param-value**>  
 </**init-param**>  
 *<!-- 设置spring容器filter的bean id，如果不设置则找与filter-name一致的bean-->* <**init-param**>  
 <**param-name**>targetBeanName</**param-name**>  
 <**param-value**>shiroFilter</**param-value**>  
 </**init-param**>  
</**filter**>  
<**filter-mapping**>  
 <**filter-name**>shiroFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
</**filter-mapping**>

## applicationContext-shiro.xml

在applicationContext-shiro.xml 中配置web.xml中fitler对应spring容器中的bean。



## 静态资源

对静态资源设置逆名访问：

修改applicationContext-shiro.xml:

*<!-- 对静态资源设置匿名访问 -->*/images/\*\* = anon  
/js/\*\* = anon  
/styles/\*\* = anon

## 登陆

### 原理

使用FormAuthenticationFilter过虑器实现 ，原理如下：

将用户没有认证时，请求loginurl进行认证，用户身份和用户密码提交数据到loginurl

FormAuthenticationFilter拦截住取出request中的username和password（两个参数名称是可以配置的）

FormAuthenticationFilter调用realm传入一个token（username和password）

realm认证时根据username查询用户信息（在Activeuser中存储，包括userid、usercode、username、menus）。

如果查询不到，realm返回null，FormAuthenticationFilter向request域中填充一个参数（记录了异常信息）

### 登陆页面

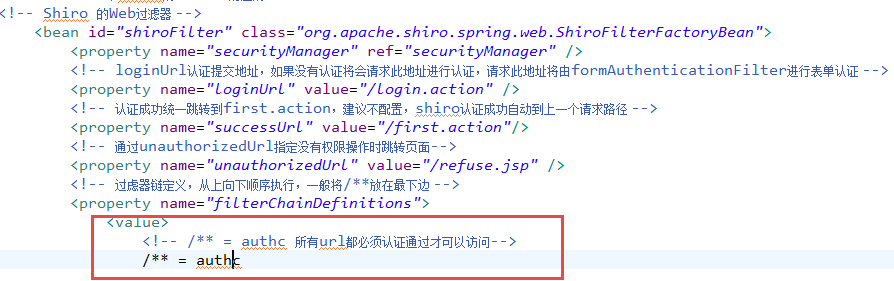
由于FormAuthenticationFilter的用户身份和密码的input的默认值（username和password），修改页面的账号和密码的input的名称为username和password

### 登陆代码实现

*//登陆提交地址，和applicationContext-shiro.xml中配置的loginurl一致*@RequestMapping(**"/login"**)  
**public** String login(HttpServletRequest request) **throws** Exception {  
 *//如果登陆失败从request中获取认证异常信息，shiroLoginFailure就是shiro异常类的全限定名* String exceptionClassName = (String) request.getAttribute(**"shiroLoginFailure"**);  
 *//根据shiro返回的异常类路径判断，抛出指定异常信息* **if** (exceptionClassName != **null**) {  
 **if** (UnknownAccountException.**class**.getName().equals(exceptionClassName)) {  
 *//最终会抛给异常处理器* **throw new** CustomException(**"账号不存在"**);  
 } **else if** (IncorrectCredentialsException.**class**.getName().equals(exceptionClassName)) {  
 **throw new** CustomException(**"密码错误"**);  
 } **else** {  
 **throw new** Exception();*//最终在异常处理器生成未知错误* }  
 }  
  
 *//此方法不处理登陆成功（认证成功），shiro认证成功会自动跳转到上一个请求路径  
 //登陆失败还到login页面* **return "user/login"**;  
}

### 认证拦截过虑器

在applicationContext-shiro.xml中配置：



## 退出

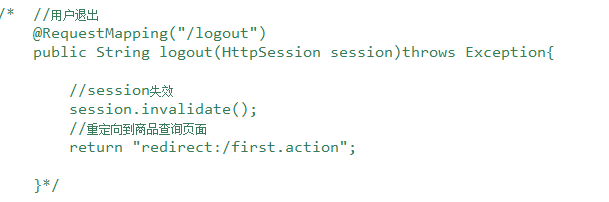
### 使用LogoutFilter

不用我们去实现退出，只要去访问一个退出的url（该 url是可以不存在），由LogoutFilter拦截住，清除session。

在applicationContext-shiro.xml配置LogoutFilter：

*<!-- 过虑器链定义，从上向下顺序执行，一般将/\*\*放在最下边 -->*<**property name="filterChainDefinitions"**>  
 <**value**>  
 *<!-- 请求 logout.action地址，shiro去清除session-->* /user/logout = logout

可以删除原来的logout的controller方法代码。



## 认证信息在页面显示

1、认证后用户菜单在首页显示

2、认证后用户的信息在页头显示

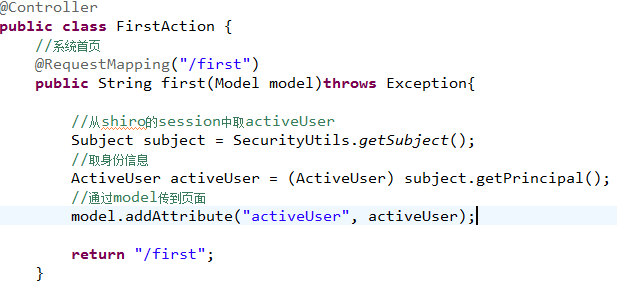
### 修改realm设置完整认证信息

realm从数据库查询用户信息，将用户菜单、usercode、username等设置在SimpleAuthenticationInfo中。

先使用静态代码实现：



### 修改first.action将认证信息在页面显示



## 授权过虑器测试

### 使用PermissionsAuthorizationFilter

在applicationContext-shiro.xml中配置url所对应的权限。

测试流程：

1、在applicationContext-shiro.xml中配置filter规则

*<!--商品查询需要商品查询权限 ，取消url拦截配置，使用注解授权方式 -->*/product/query = perms[product:query]

2、用户在认证通过后，请求/product/query

3、被PermissionsAuthorizationFilter拦截，发现需要“product:query”权限

4、PermissionsAuthorizationFilter调用realm中的doGetAuthorizationInfo获取数据库中正确的权限

5、PermissionsAuthorizationFilter对product:query和从realm中获取权限进行对比，如果“product:query”在realm返回的权限列表中，授权通过。

### 创建refuse.jsp

如果授权失败，跳转到refuse.jsp，需要在spring容器中配置：

*<!-- web.xml中shiro的filter对应的bean -->  
<!-- Shiro 的Web过滤器 -->*<**bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean"**>  
 <**property name="securityManager" ref="securityManager"**/>  
 *<!-- loginUrl认证提交地址，如果没有认证将会请求此地址进行认证，请求此地址将由formAuthenticationFilter进行表单认证 -->* <**property name="loginUrl" value="/user/login"**/>  
 *<!-- 认证成功统一跳转到first.action，建议不配置，shiro认证成功自动到上一个请求路径 -->* <**property name="successUrl" value="/first"**/>  
 *<!-- 通过unauthorizedUrl指定没有权限操作时跳转页面-->* <**property name="unauthorizedUrl" value="/refuse.jsp"**/>

### 问题总结

1、在applicationContext-shiro.xml中配置过虑器链接，需要将全部的url和权限对应起来进行配置，比较发麻不方便使用。

2、每次授权都需要调用realm查询数据库，对于系统性能有很大影响，可以通过shiro缓存来解决。

## shiro的过虑器

|  |  |
| --- | --- |
| 过滤器简称 | 对应的java类 |
| anon | org.apache.shiro.web.filter.authc.AnonymousFilter |
| authc | org.apache.shiro.web.filter.authc.FormAuthenticationFilter |
| authcBasic | org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter |
| perms | org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter |
| port | org.apache.shiro.web.filter.authz.PortFilter |
| rest | org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter |
| roles | org.apache.shiro.web.filter.authz.RolesAuthorizationFilter |
| ssl | org.apache.shiro.web.filter.authz.SslFilter |
| user | org.apache.shiro.web.filter.authc.UserFilter |
| logout | org.apache.shiro.web.filter.authc.LogoutFilter |

anon:例子/admins/\*\*=anon 没有参数，表示可以匿名使用。

authc:例如/admins/user/\*\*=authc表示需要认证(登录)才能使用，FormAuthenticationFilter是表单认证，没有参数

perms：例子/admins/user/\*\*=perms[user:add:\*],参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，例如/admins/user/\*\*=perms["user:add:\*,user:modify:\*"]，当有多个参数时必须每个参数都通过才通过，想当于isPermitedAll()方法。

user:例如/admins/user/\*\*=user没有参数表示必须存在用户, 身份认证通过或通过记住我认证通过的可以访问，当登入操作时不做检查

## 认证

### 需求

修改realm的doGetAuthenticationInfo，从数据库查询用户信息，realm返回的用户信息中包括 （md5加密后的串和salt），实现让shiro进行散列串的校验。

### 修改doGetAuthenticationInfo从数据库查询用户信息

1、将SysService注入到realm中。

**public class** CustomRealm **extends** AuthorizingRealm {  
  
 @Resource  
 **private** SysService **sysService**;

*// 用于认证*@Override  
**protected** AuthenticationInfo doGetAuthenticationInfo(  
 AuthenticationToken token) **throws** AuthenticationException {  
  
 *// token是用户输入的  
 // 第一步从token中取出身份信息 账号* String userCode = (String) token.getPrincipal();  
  
 */// 第二步：根据用户输入的userCode从数据库查询* SysUser sysUser = **null**;  
 **try** {  
 sysUser = **sysService**.findSysUserByUserCode(userCode);  
 } **catch** (Exception ignored) {  
 }  
  
 *// 如果查询不到返回null* **if** (sysUser == **null**) {  
 **return null**;  
 }  
  
 *// 从数据库查询到密码* String password = sysUser.getPassword();  
  
 *//盐* String salt = sysUser.getSalt();  
  
  
 *// 如果查询到返回认证信息AuthenticationInfo  
  
 //activeUser就是用户身份信息* ActiveUser activeUser = **new** ActiveUser();  
  
 activeUser.setUserId(sysUser.getId());  
 activeUser.setUserCode(sysUser.getUserCode());  
 activeUser.setUsername(sysUser.getUsername());  
 *//..  
  
 //根据用户id取出菜单* List<SysPermission> menus = **null**;  
 **try** {  
 *//通过service取出菜单* menus = **sysService**.findMenuListByUserId(sysUser.getId());  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 *//将用户菜单 设置到activeUser* activeUser.setMenus(menus);  
  
  
 *//将activeUser设置simpleAuthenticationInfo* SimpleAuthenticationInfo simpleAuthenticationInfo = **new** SimpleAuthenticationInfo(  
 activeUser, password, ByteSource.Util.*bytes*(salt), **this**.getName());  
  
 **return** simpleAuthenticationInfo;  
}

### 设置凭证匹配器

数据库中存储到的md5的散列值，在realm中需要设置数据库中的散列值它使用散列算法及散列次数，让shiro进行散列对比时和原始数据库中的散列值使用的算法一致。

*<!-- shiro -->*<**bean id="customRealm" class="cn.qdgxy.shiro.shiro.CustomRealm"**>  
 *<!-- 将凭证匹配器设置到realm中，realm按照凭证匹配器的要求进行散列 -->* <**property name="credentialsMatcher" ref="credentialsMatcher"**/>  
</**bean**>  
  
*<!-- 凭证匹配器 -->*<**bean id="credentialsMatcher" class="org.apache.shiro.authc.credential.HashedCredentialsMatcher"**>  
 *<!--散列算法-->* <**property name="hashAlgorithmName" value="md5"**/>  
 *<!--散列次数-->* <**property name="hashIterations" value="2"**/>  
</**bean**>

## 授权

### 需求

修改realm的doGetAuthorizationInfo，从数据库查询权限信息。

使用注解式授权方法。

使用jsp标签授权方法。

### 修改doGetAuthorizationInfo从数据库查询权限

*// 用于授权*@Override  
**protected** AuthorizationInfo doGetAuthorizationInfo(  
 PrincipalCollection principals) {  
  
 *//从 principals获取主身份信息  
 //将getPrimaryPrincipal方法返回值转为真实身份类型（在上边的doGetAuthenticationInfo认证通过填充到SimpleAuthenticationInfo中身份类型），* ActiveUser activeUser = (ActiveUser) principals.getPrimaryPrincipal();  
  
 *//根据身份信息获取权限信息  
 //从数据库获取到权限数据* List<SysPermission> permissionList = **null**;  
 **try** {  
 permissionList = **sysService**.findPermissionListByUserId(activeUser.getUserId());  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 *//单独定一个集合对象* List<String> permissions = **new** ArrayList<>();  
 **if** (permissionList != **null**) {  
 **for** (SysPermission sysPermission : permissionList) {  
 *//将数据库中的权限标签 符放入集合* permissions.add(sysPermission.getPercode());  
 }  
 }  
  
 *//查到权限数据，返回授权信息(要包括上边的permissions)* SimpleAuthorizationInfo simpleAuthorizationInfo = **new** SimpleAuthorizationInfo();  
 *//将上边查询到授权信息填充到simpleAuthorizationInfo对象中* simpleAuthorizationInfo.addStringPermissions(permissions);  
  
 **return** simpleAuthorizationInfo;  
}

### 开启controller类aop支持

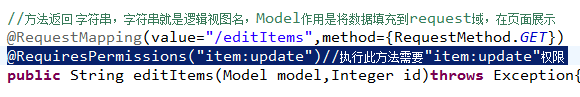
对系统中类的方法给用户授权，建议在controller层进行方法授权。

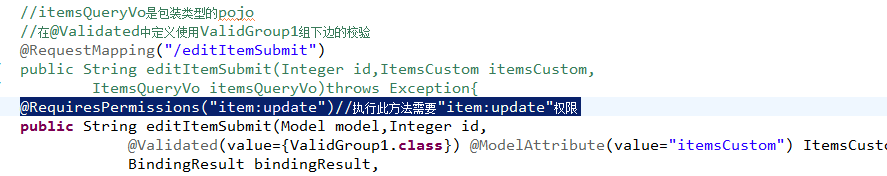
在springmvc.xml中配置：



### 在controller方法中添加注解

@RequestMapping(**"/all"**)  
@RequiresPermissions(**"product:queryAll"**)*//执行queryItems需要"product:queryAll"权限***public** ModelAndView findAllProduct(HttpServletRequest request) **throws** Exception {





### jsp标签 授权

Jsp页面添加：

<%@ **taglib prefix**="**shiro**" **uri**="**http://shiro.apache.org/tags**" %>

|  |  |
| --- | --- |
| 标签名称 | 标签条件（均是显示标签内容） |
| <shiro:authenticated> | 登录之后 |
| <shiro:notAuthenticated> | 不在登录状态时 |
| <shiro:guest> | 用户在没有RememberMe时 |
| <shiro:user> | 用户在RememberMe时 |
| <shiro:hasAnyRoles name="abc,123" > | 在有abc或者123角色时 |
| <shiro:hasRole name="abc"> | 拥有角色abc |
| <shiro:lacksRole name="abc"> | 没有角色abc |
| <shiro:hasPermission name="abc"> | 拥有权限资源abc |
| <shiro:lacksPermission name="abc"> | 没有abc权限资源 |
| <shiro:principal> | 显示用户身份名称 |

<shiro:principal property="username"/> 显示用户身份中的属性值

修改itemsList.jsp页面：

*<!-- 有product:update权限才显示修改链接，没有该 权限不显示，相当 于if(hasPermission(product:update)) -->*<**shiro:hasPermission name="product:update"**>  
 <**td**>  
 <**a href="${**pageContext.request.contextPath **}/product/editItem.action?id=${**product.id**}"**>修改</**a**>  
 </**td**>  
</**shiro:hasPermission**>

### shiro标签 授权

第一步：下载shiro-freemarker-tags GitHub地址：https://github.com/jagregory/shiro-freemarker-tags   
可以先看一下他的说明文档，有一个初步的了解。  
  
第二步：Spring MVC配置  
自定义一个ShiroTagFreeMarkerConfigurer继承Spring本身提供的FreeMarkerConfigurer,目的是在FreeMarker的Configuration中添加shiro的配置

**public class** ShiroTagFreeMarkerConfigurer **extends** FreeMarkerConfigurer {  
  
 @Override  
 **public void** afterPropertiesSet() **throws** IOException, TemplateException {  
 **super**.afterPropertiesSet();  
 **this**.getConfiguration().setSharedVariable(**"shiro"**, **new** ShiroTags());  
 }  
  
}

第三步：xml文件配置

<**bean id="freemarkerConfig" class="cn.qdgxy.tr.web.shiro.ShiroTagFreeMarkerConfigurer"**>  
 <**property name="templateLoaderPath" value="/WEB-INF/ftl/"**/>  
 <**property name="defaultEncoding" value="utf-8"**/>  
 <**property name="freemarkerSettings"**>  
 <**props**>  
 <**prop key="template\_update\_delay"**>10</**prop**>  
 <**prop key="locale"**>zh\_CN</**prop**>  
 <**prop key="datetime\_format"**>yyyy-MM-dd HH:mm:ss</**prop**>  
 <**prop key="date\_format"**>yyyy-MM-dd HH:mm:ss</**prop**>  
 <**prop key="number\_format"**>#.##</**prop**>  
 </**props**>  
 </**property**>  
</**bean**>

第四步：使用Shiro Tag

**<@shiro**.**guest>**Hello guest!**</@shiro**.**guest>**

### 授权测试

当调用controller的一个方法，由于该方法加了@RequiresPermissions("product:update") ，shiro调用realm获取数据库中的权限信息，看"product:update"是否在权限数据中存在，如果不存在就拒绝访问，如果存在就授权通过。

当展示一个jsp页面时，页面中如果遇到<shiro:hasPermission name="item:update">，shiro调用realm获取数据库中的权限信息，看item:update是否在权限数据中存在，如果不存在就拒绝访问，如果存在就授权通过。

问题：只要遇到注解或jsp标签的授权，都会调用realm方法查询数据库，需要使用缓存解决此问题。

## shiro缓存

针对上边授权频繁查询数据库，需要使用shiro缓存。

### 缓存流程

shiro中提供了对认证信息和授权信息的缓存。shiro默认是关闭认证信息缓存的，对于授权信息的缓存shiro默认开启的。主要研究授权信息缓存，因为授权的数据量大。

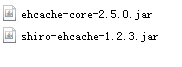
用户认证通过。

该用户第一次授权：调用realm查询数据库

该用户第二次授权：不调用realm查询数据库，直接从缓存中取出授权信息（权限标识符）。

### 使用ehcache

#### 添加Ehcache的jar包



#### 配置cacheManager

*<!-- securityManager安全管理器 -->*<**bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager"**>  
 <**property name="realm" ref="customRealm"**/>  
 *<!-- 注入缓存管理器 -->* <**property name="cacheManager" ref="cacheManager"**/></**bean**>

*<!-- 缓存管理器 -->*<**bean id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager"**>  
 <**property name="cacheManagerConfigFile" value="classpath:shiro-ehcache.xml"**/>  
</**bean**>

#### shiro-ehcache.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:noNamespaceSchemaLocation="ehcache.xsd"  
 updateCheck="true"  
 monitoring="autodetect"  
 dynamicConfig="true"**>  
  
 *<!-- 硬盘中的临时存放目录 -->* <**diskStore path="F:/EHCache"**/>  
  
 *<!--   
 name ：cache唯一标识  
 maxElementsInMemory ：内存中最大缓存对象数  
 eternal ：缓存是否永久有效  
 overflowToDisk(true,false)：缓存对象达到最大数后，将缓存写到硬盘中   
 diskPersistent ：硬盘持久化   
 timeToIdleSeconds ：缓存清除时间   
 timeToLiveSeconds ：缓存存活时间   
 memoryStoreEvictionPolicy ：缓存清空策略   
 1. FIFO ：first in first out 先讲先出   
 2. LFU ： Less Frequently Used 一直以来最少被使用的   
 3. LRU ：Least Recently Used 最近最少使用的   
 -->* <**defaultCache  
 maxElementsInMemory="1000"  
 maxElementsOnDisk="10000000"  
 eternal="false"  
 overflowToDisk="false"  
 diskPersistent="false"  
 timeToIdleSeconds="120"  
 timeToLiveSeconds="120"  
 diskExpiryThreadIntervalSeconds="120"  
 memoryStoreEvictionPolicy="LRU"**>  
 </**defaultCache**>  
</**ehcache**>

#### 缓存清空

如果用户正常退出，缓存自动清空。

如果用户非正常退出，缓存自动清空。

如果修改了用户的权限，而用户不退出系统，修改的权限无法立即生效。

需要手动进行编程实现：

**在权限修改后调用realm的clearCache方法清除缓存。**

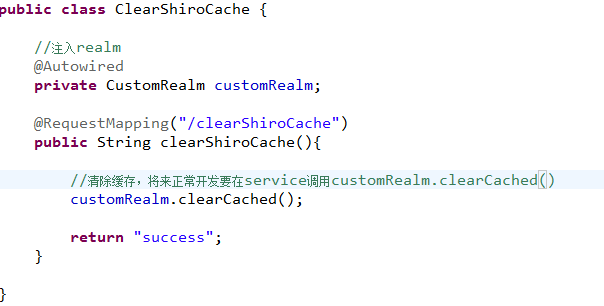
下边的代码正常开发时要放在service中调用。

在service中，权限修改后调用realm的方法。

在realm中定义clearCached方法：

*/\*\*  
 \* 清除缓存  
 \*/***public void** clearCached() {  
 PrincipalCollection principals = SecurityUtils.*getSubject*().getPrincipals();  
 **super**.clearCache(principals);  
}

测试清除缓存controller方法：



## sessionManager

和shiro整合后，使用shiro的session管理，shiro提供sessionDao操作会话数据。

配置sessionManager

*<!-- securityManager安全管理器 -->*<**bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager"**>  
 <**property name="realm" ref="customRealm"**/>  
 *<!-- 注入缓存管理器 -->* <**property name="cacheManager" ref="cacheManager"**/>  
 *<!-- 注入session管理器 -->* <**property name="sessionManager" ref="sessionManager"**/></**bean**>

*<!-- 会话管理器 -->*<**bean id="sessionManager" class="org.apache.shiro.web.session.mgt.DefaultWebSessionManager"**>  
 *<!-- session的失效时长，单位毫秒 -->* <**property name="globalSessionTimeout" value="600000"**/>  
 *<!-- 删除失效的session -->* <**property name="deleteInvalidSessions" value="true"**/>  
</**bean**>

## 验证码

### 思路

shiro使用FormAuthenticationFilter进行表单认证，验证校验的功能应该加在FormAuthenticationFilter中，在认证之前进行验证码校验。

需要写FormAuthenticationFilter的子类，继承FormAuthenticationFilter，改写它的认证方法，在认证之前进行验证码校验。

### 自定义FormAuthenticationFilter

*/\*\*  
 \* 自定义FormAuthenticationFilter，认证之前实现 验证码校验  
 \* create:2016-11-13 20:28  
 \*  
 \** ***@author*** *李欣  
 \** ***@version*** *1.0  
 \*/***public class** CustomFormAuthenticationFilter **extends** FormAuthenticationFilter {  
  
 */\*\*  
 \* 原FormAuthenticationFilter的认证方法  
 \*  
 \** ***@param request*** *request  
 \** ***@param response*** *response  
 \** ***@return*** *boolean  
 \** ***@throws*** *Exception Exception  
 \*/* @Override  
 **protected boolean** onAccessDenied(ServletRequest request,  
 ServletResponse response) **throws** Exception {  
 HttpServletRequest httpServletRequest = (HttpServletRequest) request;  
 HttpSession session = httpServletRequest.getSession();  
  
 *//在这里进行验证码的校验  
 //取出session的验证码（正确的验证码）* String validateCode = (String) session.getAttribute(**"validateCode"**);  
  
 *//取出页面的验证码  
 //输入的验证和session中的验证进行对比* String randomcode = httpServletRequest.getParameter(**"randomcode"**);  
 **if** (randomcode != **null** && validateCode != **null** && !randomcode.equalsIgnoreCase(validateCode)) {  
 *//如果校验失败，将验证码错误失败信息，通过shiroLoginFailure设置到request中* httpServletRequest.setAttribute(**"shiroLoginFailure"**, **"randomCodeError"**);  
 *//拒绝访问，不再校验账号和密码* **return true**;  
 }  
  
 **return super**.onAccessDenied(request, response);  
 }  
  
}

### 配置自定义FormAuthenticationFilter

<**bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean"**>  
 <**property name="securityManager" ref="securityManager"**/>  
 *<!-- loginUrl认证提交地址，如果没有认证将会请求此地址进行认证，请求此地址将由formAuthenticationFilter进行表单认证 -->* <**property name="loginUrl" value="/user/login"**/>  
 *<!-- 认证成功统一跳转到first.action，建议不配置，shiro认证成功自动到上一个请求路径 -->* <**property name="successUrl" value="/first"**/>  
 *<!-- 通过unauthorizedUrl指定没有权限操作时跳转页面-->* <**property name="unauthorizedUrl" value="/refuse.jsp"**/>  
 *<!-- 自定义filter配置 -->* <**property name="filters"**>  
 <**map**>  
 *<!-- 将自定义 的FormAuthenticationFilter注入shiroFilter中-->* <**entry key="authc" value-ref="formAuthenticationFilter"**/>  
 </**map**>  
 </**property**>

*<!-- 自定义form认证过虑器 -->  
<!-- 基于Form表单的身份验证过滤器，不配置将也会注册此过虑器，表单中的用户账号、密码及loginurl将采用默认值，建议配置 -->*<**bean id="formAuthenticationFilter" class="cn.qdgxy.shiro.shiro.CustomFormAuthenticationFilter"**>  
 *<!-- 表单中账号的input名称 -->* <**property name="usernameParam" value="username"**/>  
 *<!-- 表单中密码的input名称 -->* <**property name="passwordParam" value="password"**/>  
 *<!-- 记住我input的名称 -->* <**property name="rememberMeParam" value="rememberMe"**/>  
</**bean**>

### 在login.action对验证错误 进行解析

*//登陆提交地址，和applicationContext-shiro.xml中配置的loginurl一致*@RequestMapping(**"/login"**)  
**public** String login(HttpServletRequest request) **throws** Exception {  
 *//如果登陆失败从request中获取认证异常信息，shiroLoginFailure就是shiro异常类的全限定名* String exceptionClassName = (String) request.getAttribute(**"shiroLoginFailure"**);  
 *//根据shiro返回的异常类路径判断，抛出指定异常信息* **if** (exceptionClassName != **null**) {  
 **if** (UnknownAccountException.**class**.getName().equals(exceptionClassName)) {  
 *//最终会抛给异常处理器* **throw new** CustomException(**"账号不存在"**);  
 } **else if** (IncorrectCredentialsException.**class**.getName().equals(exceptionClassName)) {  
 **throw new** CustomException(**"密码错误"**);  
 } **else if** (**"randomCodeError"**.equals(exceptionClassName)) {  
 **throw new** CustomException(**"验证码错误 "**);  
 } **else** {  
 **throw new** Exception();*//最终在异常处理器生成未知错误* }  
 }  
  
 *//此方法不处理登陆成功（认证成功），shiro认证成功会自动跳转到上一个请求路径  
 //登陆失败还到login页面* **return "user/login"**;  
}

### 在登陆页面添加验证码

### 在filter配置匿名访问验证码jsp

<**TR**>  
 <**TD**>用户名：</**TD**>  
 <**TD colSpan="2"**><**input type="text" id="usercode" name="username" style="WIDTH**: 130**px"**/></**TD**>  
</**TR**>  
<**TR**>  
 <**TD**>密 码：</**TD**>  
 <**TD**><**input type="password" id="pwd" name="password" style="WIDTH**: 130**px"**/>  
 </**TD**>  
</**TR**>  
<**TR**>  
 <**TD**>验证码：</**TD**>  
 <**TD**>  
 <**input id="randomcode" name="randomcode" size="8"**/>  
 <**img id="randomcode\_img" src="${**baseurl**}/validatecode.jsp" alt="" width="56" height="20"  
 align='absMiddle'**/>  
 <**a href=javascript:randomcode\_refresh()**>刷新</**a**>  
 </**TD**>  
</**TR**>

## 记住我

用户登陆选择“自动登陆”本次登陆成功会向cookie写身份信息，下次登陆从cookie中取出身份信息实现自动登陆。

### 用户身份实现java.io.Serializable接口

向cookie记录身份信息需要用户身份信息对象实现序列化接口，如下：

*/\*\*  
 \* 用户身份信息，存入session 由于tomcat将session会序列化在本地硬盘上，所以使用Serializable接口  
 \*/***public class** ActiveUser **implements** Serializable {  
  
 **private** Long **userId**;*//用户id* **private** String **userCode**;*// 用户账号* **private** String **username**;*// 用户名称* **private** List<SysPermission> **menus**;*// 菜单* **private** List<SysPermission> **permissions**;*// 权限*

**public class** SysPermission **extends** Model<SysPermission> **implements** Serializable {

### 配置rememeberMeManager

*<!-- securityManager安全管理器 -->*<**bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager"**>  
 <**property name="realm" ref="customRealm"**/>  
 *<!-- 注入缓存管理器 -->* <**property name="cacheManager" ref="cacheManager"**/>  
 *<!-- 注入session管理器 -->* <**property name="sessionManager" ref="sessionManager"**/>  
 *<!-- 记住我 -->* <**property name="rememberMeManager" ref="rememberMeManager"**/>  
</**bean**>

*<!--rememberMeManager管理器，写cookie，取出cookie生成用户信息-->*<**bean id="rememberMeManager" class="org.apache.shiro.web.mgt.CookieRememberMeManager"**>  
 <**property name="cookie" ref="rememberMeCookie"**/>  
</**bean**>  
*<!-- 记住我cookie -->*<**bean id="rememberMeCookie" class="org.apache.shiro.web.servlet.SimpleCookie"**>  
 *<!-- rememberMe是cookie的名字 -->* <**constructor-arg value="rememberMe"**/>  
 *<!-- 记住我cookie生效时间30天，单位为秒 -->* <**property name="maxAge" value="2592000"**/>  
</**bean**>

### 登陆页面

<**tr**>  
 <**TD**></**TD**>  
 <**td**><**input type="checkbox" name="rememberMe"**/>自动登陆</**td**>  
</**tr**>

### 配置rememberMe的input名称

*<!-- 自定义form认证过虑器 -->  
<!-- 基于Form表单的身份验证过滤器，不配置将也会注册此过虑器，表单中的用户账号、密码及loginurl将采用默认值，建议配置 -->*<**bean id="formAuthenticationFilter" class="cn.qdgxy.shiro.shiro.CustomFormAuthenticationFilter"**>  
 *<!-- 表单中账号的input名称 -->* <**property name="usernameParam" value="username"**/>  
 *<!-- 表单中密码的input名称 -->* <**property name="passwordParam" value="password"**/>  
 *<!-- 记住我input的名称 -->* <**property name="rememberMeParam" value="rememberMe"**/>  
</**bean**>

### 测试

自动登陆后，需要查看 cookei是否有rememberMe



### 使用UserFilter

如果设置记住我，下次访问某些url时可以不用登陆。将记住我即可访问的地址配置让UserFilter拦截。

*<!-- 过虑器链定义，从上向下顺序执行，一般将/\*\*放在最下边 -->*<**property name="filterChainDefinitions"**>  
 <**value**>  
 *<!-- 对静态资源设置匿名访问 -->* /images/\*\* = anon  
 /js/\*\* = anon  
 /styles/\*\* = anon  
  
 *<!-- 验证码，可匿名访问 -->* /validatecode.jsp = anon  
  
 *<!-- 请求 logout.action地址，shiro去清除session-->* /user/logout = logout  
  
 *<!--商品查询需要商品查询权限 ，取消url拦截配置，使用注解授权方式 -->  
 <!--/product/all = perms[product:queryAll]-->  
  
 <!-- /items/editItems.action = perms[item:edit] -->  
 <!-- 配置记住我或认证通过可以访问的地址 -->* / = user  
 /first = user  
 /welcome.jsp = user  
  
 *<!-- /\*\* = authc 所有url都必须认证通过才可以访问-->* /\*\* = authc  
  
 *<!-- /\*\* = anon 所有url都可以匿名访问 -->  
 <!--/\*\* = anon-->* </**value**>  
</**property**>